MASSACHUSETTS
INSTITUTE of TECHNOLOGY

HD28
.M414
no.1939-87

DEWEY
M.I.T. INST. TECH.
NOV 12 1987
LIBRARIES

WORKING PAPER
ALFRED P. SLOAN SCHOOL OF MANAGEMENT

Toshiba's Fuchu Software Factory:

Strategy, Technology, and Organization

Michael A. Cusumano

October 6, 1987 WP #1939-87

MASSACHUSETTS
INSTITUTE OF TECHNOLOGY
50 MEMORIAL DRIVE
CAMBRIDGE, MASSACHUSETTS 02139

Toshiba's Fuchu Software Factory:

Strategy, Technology, and Organization

Michael A. Cusumano

October 6, 1987 WP #1939-87

M.I.T. LIBRARIES
NOV 1 2 1987
RECEIVED

Michael A. Cusumano
MIT Sloan School of Management

10/6/87
Software Project Paper #4

TOSHIBA'S FUCHU SOFTWARE FACTORY: STRATEGY, TECHNOLOGY, AND ORGANIZATION

INTRODUCTION

This paper is part of a larger study examining the question of whether or not companies are choosing to manage a complex engineering activity such as large-scale software development with a range of strategic considerations and organizational as well as technological approaches that corresponds to the spectrum usually associated with "hard" manufacturing, i.e. job shops, batch organizations, and factories exhibiting various degrees of flexibility in product mixes and technologies. The research project includes the proposal of technology and policy criteria defining what a factory environment for software might look like; a survey of major software facilities in the U.S. and Japan to determine where firms stand in relation to these criteria; and detailed case studies examining the technology and policy implementation process followed at firms identified as being close to the factory model.[1]

There are several interrelated conclusions: (1) This spectrum, including "factory" approaches, is clearly observable in the sample of software facilities in the U.S. and Japan. (2) There appears to be nothing inherent in software as a technology that prevents some firms from creating strategies and organizational structures to manage product and process development more effectively, even with a relatively new and complex technology such as software. (3) The basic technological infrastructures to aid software process

management are not significantly different between Japanese and U.S. firms. (4) But, Japanese firms -- led by the NEC group and Toshiba, and followed by Hitachi and Fujitsu -- are significantly ahead of most U.S. competitors in implementing "flexible factory" type of strategies focused on reusing standardized components (modules of code) and then customizing end products.

This paper extends the survey approach to analyze what is probably the most difficult aspect of the software factory -- the implementation process and the benefits or disadvantages this environment might offer in operation. The Toshiba case is significant for several reasons.

One, like Hitachi and NEC, market demands and a shortage of skilled personnel influenced Toshiba managers to attempt to rationalize software development alone the lines of a factory organization focused on reusing code to produce semi-customized programs. The center for developing software engineering technology was not Toshiba's computer division, however, but its heavy industrial equipment division. A senior engineer, Dr. Matsumoto Yoshihiro, who originally established the factory, was especially influenced by the SDC Software Factory experiment and cited SDC in a 1981 article to justify his own efforts.[2] In 1977, while the SDC experiment was still in operation, Matsumoto directed the founding of a Toshiba Software Factory within the company's Fuchu Works, which manufactured hardware and software systems for nuclear power plants, electric power plants, factory automation, as well as elevators and transportation equipment.

Second, rather than end the effort midway when difficulties arose, as occurred at SDC, or display a reluctance to overemphasize the factory paradigm and reusability, as at Hitachi or Fujitsu, Matsumoto continued to

develop the facility's technological and management systems to support the objective of producing standardized components in a standardized format. The Toshiba factory in 1987, with 2300 programmers, was probably the largest dedicated software facility in the world, with highly focused products and remarkably rationalized factory procedures, especially in the areas of testing and the generation and reuse of standardized software components.

A third significant aspect of this case is that Toshiba has transferred the factory tool set and development approach to software product areas outside of industrial control applications. Success in this transfer reflect the broad applicability of the strategy, technology, and management systems Toshiba has devised for large-scale software engineering.

I. ORIGINS AND ORGANIZATION OF THE FUCHU SOFTWARE FACTORY

Corporate Setting

Toshiba's founding dates back to 1875 and the establishment of a small firm that manufactured communications equipment. In 1904 the firm became affiliated with the Mitsui group in Japan and in 1910 with General Electric of the U.S. Both relationships continued into the 1980s, by which time Toshiba had grown to become Japan's second largest general electrical equipment and appliance manufacturer, with 70,000 employees (excluding subsidiaries). Total worldwide sales were about $20 billion in 1986, divided mainly among industrial electronics and electronic components, including computers and office automation (33%); consumer products, including audio, video, and household appliances (31%); heavy electrical apparatus, including

power plant systems, industrial equipment, and factory automation systems (26%).[3] Among Japan's domestic computer manufacturers, Toshiba ranked fourth, behind Fujitsu, NEC, and Hitachi, although it was the leading Japanese producer of minicomputers and a major producer of office equipment.[4] The company used to be a comprehensive manufacturer of computers, ranging from small machines to mainframes, but withdrew from mainframes in 1978, selling this business to its development partner, NEC.

In 1987, there were approximately 11,000 software engineers and programmers in Toshiba. These were located in Fuchu, which alone had 2300 programmers and designers in the Software Factory plus another 1000 developing microcomputer software, at six smaller software factories (see Table 1), and in various MIS (management-information systems) activities throughout the corporation. Most tool and method development was done at the corporate R&D level and the Fuchu Plant.[5]

Table 1: TOSHIBA SOFTWARE FACTORIES

Facility	Software Product Area
Fuchu	Real-Time Industrial Applications
Ome	Office Automation
Hino	Telecommunications
Komukai	Defense Systems (radar, satellites)
Yanagimachi	Automatic Dispensing Machines (banks, railway tickets)
Nasu	Medical Systems
Isogo	Home Appliances

Source: Matsumoto interview.

The Factory Strategy

The establishment of the Fuchu Software Factory was directly connected with the characteristics of Toshiba's industrial control systems business, which dates back to the use of electro-magnetic relays prior to the use of transistors for this application from the mid-1960s. Software development became a large area of activity after the introduction of the 4004 microprocessor in 1971 and the Fuchu Works's first control system using a microprocessor in 1974. Major functions of industrial control systems that involve extensive software writing are processing the relevant data to control the production equipment; serving as an interface between human operators and the machinery; and providing sufficient information on production levels, inventories, plant conditions, etc., to serve as an interface for management.[6]

According to Matsumoto, it was primarily a shortage software programmers to meet the demand to Toshiba for customized programs caused by the popularity of "off-the-shelf" process-control minicomputers that prompted the idea of establishing a software factory focused on productivity improvement. Orders for these computers to Toshiba began increasing rapidly from 1977. Regarding ways to rationalize software development using factory approaches and work bench systems, Matsumoto was also very much influenced by the 1975 article in **Computer** on the System Development Corporation's Software Factory experiment, as well as by an article in the mid-1970s on the Programmers' Work Bench system (PWB) developed at AT&T, which used the Unix operating system. The name of Toshiba's SWB system Matsumoto patterned after the AT&T system.[7]

Quality control was secondary but still important motivation behind the

establishment of the factory. Customers were mainly sensitive to quality, according to Matsumoto, and did not directly see the impact of productivity improvements in software development. The cost of the software was decided along with the hardware and not specifically separated for most projects. On the other hand, Toshiba internally was directly affected by productivity because of the impact this had on its capacity for handling software orders.[8]

A typical example of the problem Toshiba faced was an order to develop the hardware and software for what would be the world's first fully automated thermal power generating station for Tokyo Electric, the Hirano Plant. Not only did this require software running into several millions of lines of code. To achieve safe and untended operation, hardware and software reliability requirements were extremely stringent, placing tremendous demand's on Toshiba software capabilities.

Hirano presented problems in productivity and quality assurance that became typical of Toshiba projects as its sales of minicomputers increased. As Matsumoto noted in a 1981 article, since the Software Factory's main products are real-time systems for facilities such as nuclear power plants, electric utilities networks, chemical processing factories, air terminals, and steel rolling mills, failures would be "disastrous." Thus, they decided to build an entire facility dedicated to maximizing quality assurance of the product at a price that still guaranteed Toshiba "a moderate profit."[9] The Fuchu Software Factory opened in 1977, prior to the completion of the Tokyo Electric Hirano project.[10]

As a strategy to improve productivity, quality, and cost control for the Fuchu Work's entire software needs, management used the set of tools and

procedures developed under the Software Factory rubric to provide a standardized development environment for the three primary applications areas Fuchu served: heavy electrical equipment, measuring instruments, and industrial control systems. Within a few years, Toshiba managers recognized that the factory concept should provide useful to produce other types of software relatively inexpensively and with high quality. By the early 1980s, the factory's design activities were extended to a broad range of process-control systems, factory automation, and measuring equipment software.[11]

The philosophy underlying the factory organization, including its tools and procedures, can be seen in Matsumoto's comments in a 1984 article in IEEE Computer. The notion of "factory production" as different from software "development" lay in a strategic commitment, and supporting technological and policy infrastructures, to producing software by reusing as many existing components as possible. The way to achieve this was eclectic, using any type of concepts or tools that furthered this goal. Quality, on the other hand, was seen as linked directly to human performance, and in this aspect good "people management" as well as technology management were considered essential to effective operation of the factory:

> Software production is distinct from software development in that production management is directed toward the use of existing software and enforces the idea that designers should build new software from existing codes. The software life-cycle model and various software engineering techniques based on the model are therefore worthwhile from the viewpoint of production management. However, other development systems deviate from the life-cycle approach, such as prototyping and program generating, that are equally important and vital to the development of new software. The concept of 'levels of abstraction' is applied to both early design and manufacturing phases in order to incorporate these techniques.
>
> In order to manage traces between different abstract levels, modularity or encapsulation in the requirements and design levels is considered. Modularity in the early stages of software development aids in the reuse of existing modules and prototyping. However, human factors in

> software management are becoming increasingly important; therefore human-oriented reviews and inspections are being applied to increase software reliability.[12]

Another central notion of the factory was that product quality should not be dependent on individual performance. This was standard in most factory approaches to quality control. What distinguished the software factory approach at Toshiba, however, was the commitment to using technological tools to support all aspects of software development and production, and to improve quality through compensating for differences in the ability of individual engineers, as Matsumoto noted in a 1986 paper:

> The path from requirements definition to conceptual design primarily requires the intellectual ability of the engineer. Supporting this part is believed to produce the greatest achievements in quality improvement by **leveling off the engineers' ability** [my italics]. Support for this part, however, was considered technically difficult. Fortunately, the recent progress in work stations and knowledge engineering helped to solve this problem. As a result, ... support elements are added to the SWB system.[13]

For example, one of the tools Toshiba developed was a "lexical editor," which, with a simple command, automatically displayed constructions such as "do while" for different languages. Programmers then simply filled in different parameters, without having to study the grammar of different languages and risk making coding errors.[14]

According to Kado Tadao, in 1982 the manager of the engineering administration department at the Fuchu Plant, due to improvements in quality and rationalization of "asset management," the volume of work the factory has been able to handle in 1986 was triple that of the original plan. Most important to Kado in this long-term productivity improvement were (1)

selection of good quality software engineers; (2) development of good tools and methods; and (3) creation of a good working environment and training system.[15]

The Factory Structure

Matsumoto defined a "software factory" as "an environment which allows software manufacturing organizations to design, program, test, ship, install, and maintain commercial software products in a unified manner...[and] attain specified quality and productivity levels." Apart from the formal departmental divisions, management implemented its strategy through the combination of 14 elements or systems.[16] These can be broken down into tool/facility infrastructural elements and management-personnel systems, which will be elaborated on in the subsequent sections of this paper (Table 2):

Table 2: ELEMENTS OF THE TOSHIBA SOFTWARE FACTORY

Tool/Facility Infrastructure

-- Specially designed work spaces.

-- Software tools, user interfaces and tool maintenance facilities.

-- Existing software library and maintenance support for this.

-- Technical data library.

-- Standardized technical methodologies and disciplines.

-- Documentation support.

Management/Personnel Systems

-- Project progress management system.

-- Cost management system.

-- Productivity management system.

-- Quality assurance system with standardized quality metrics.

-- A standardized, baseline management system for design review, inspection and configuration management.

-- Education programs.

-- Quality circle activities.

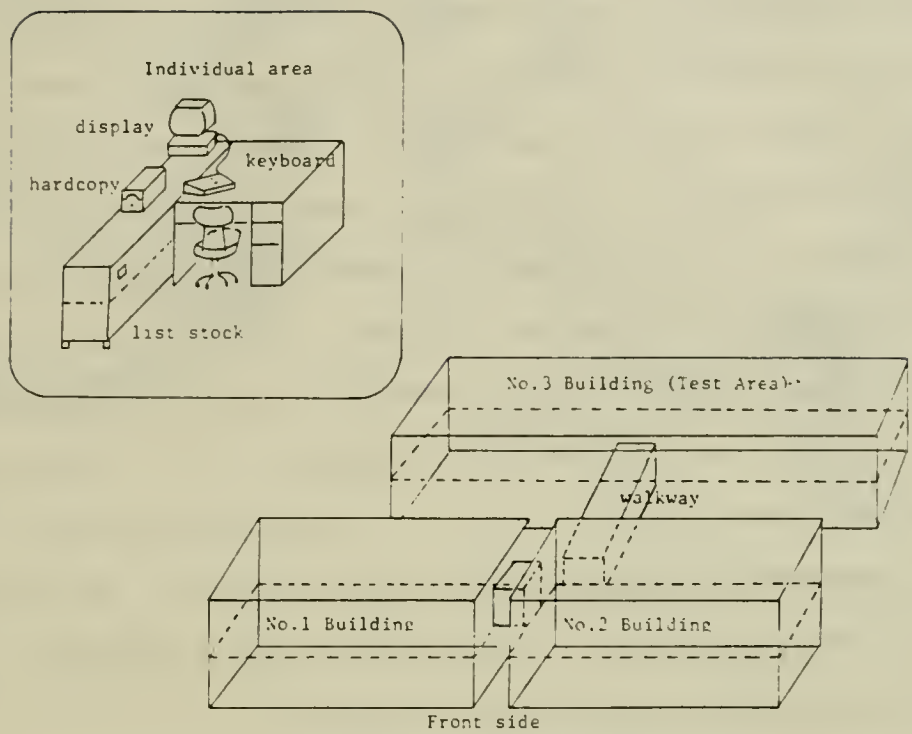
-- Career development system.

Source: Yoshihiro Matsumoto (Toshiba Corporation), "A Software Factory: An Overall Approach to Software Production," in Peter Freeman, ed., Software Reusability (IEEE Tutorial, 1987), p. 1 (2 December 1986 manuscript).

The Software Factory employees performed some system engineering but focused on detailed design, programming, testing, quality assurance, project management, installation, plant-site alignment and maintenance. Volume,

measured by software shipments to customers, totaled about 7.2 million equivalent assembler source lines (EASL) per month in 1986-1987 (about 1.3 million lines of source code), excluding basic software such as operating systems, utilities, and language processors. The average size of an application software project was 4 million EASL (about 700,000 lines of source code), and the range was 1 to 21 EASL. Projects generally consisted of 150 to 300 real-time tasks and took more than three years to complete. Systems sold to customers usually included, in addition to computer and peripherals hardware, the application software and subsidiary application packages, a utility subsystem (database management systems, user interfaces, input-output interfaces), and an operating system.[17]

The facilities in 1987 consisted of three interconnected buildings (Figure 1) with a combined floor space of 17,000 square meters. The first two contained the terminals or work stations, referred to as the Software Work Bench (SWB) facilities, individual work areas, an SWB service center, a documentation service center, a file storage stack room, and a lounge. The third building (in the rear) contained facilities for testing completed software with hardware manufactured by other departments. Each individual work area had an SWB terminal and CRT display, a key board, and a small hard-copy printer.[18] Space and money constraints limited the factory to approximately one terminal per four programmers, compared to one each for programmers at U.S. firms such as Hughes Aircraft. Toshiba had plans to erect another building and increase the number of terminals, as well as to add lap-top computers to serve as terminals.[19]



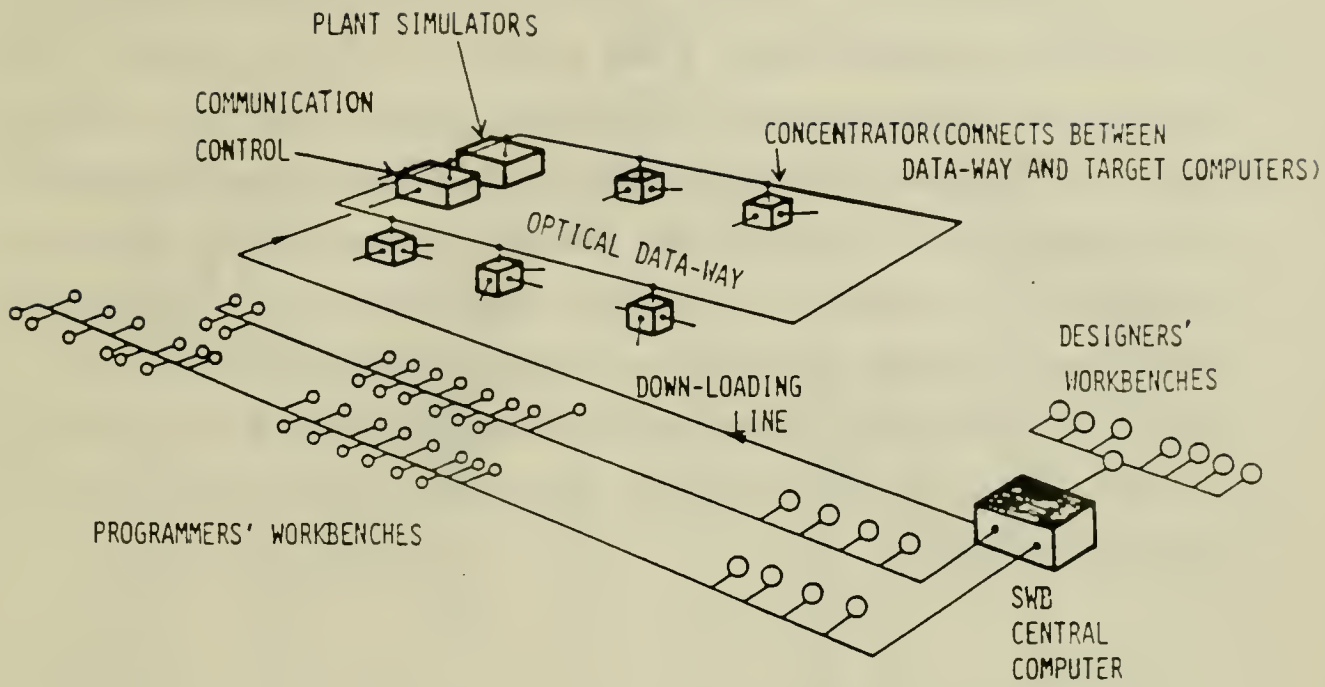
An overhead view of the software factory

A view of the software factory from front side

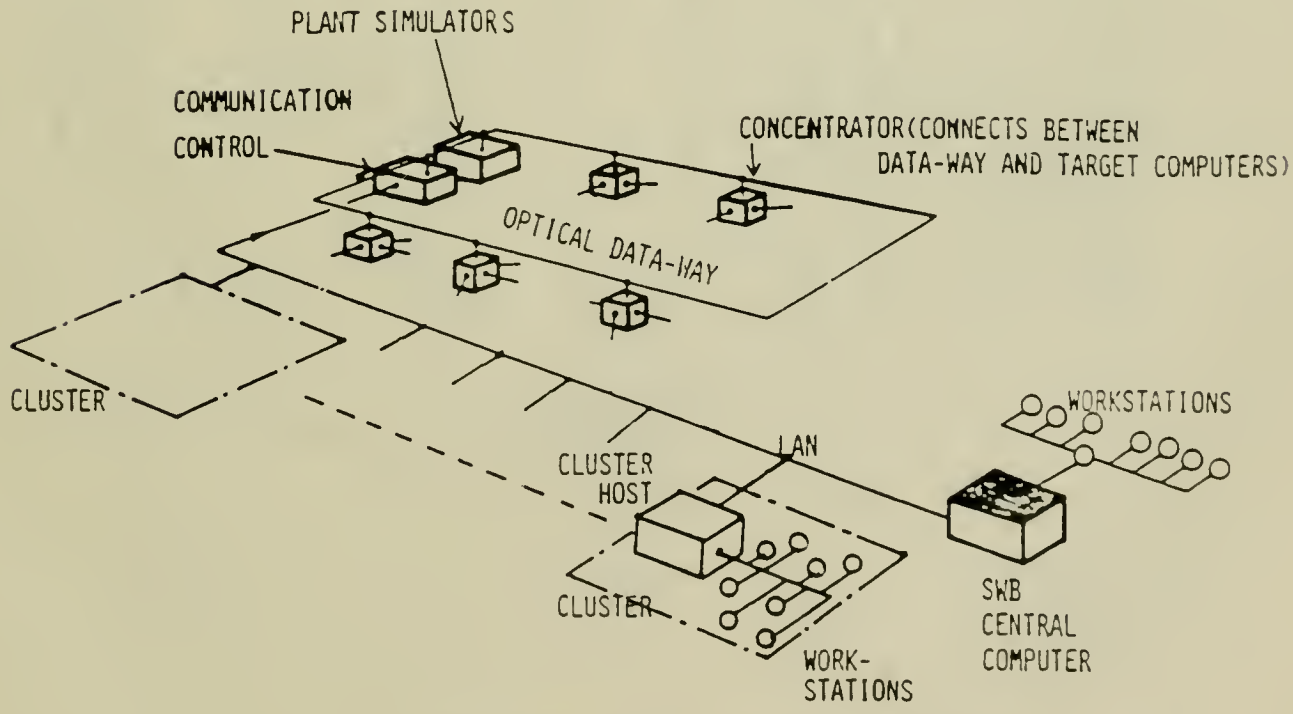
Fig. 1

The initial development staff consisted of 10 to 15 members at the beginning in 1977.[20] Funding for development of the technological infrastructure (the SWB system) came not from the Hirano project or even the Fuchu Works, but from corporate funds.[21] The SWB system was actually first called SPS (Software Production System), and was designed to provide a single uniform interface and general centralized support for software development by programmers located at distributed sites (not a single location). Introduction of the system beginning in October 1977 was at six different Toshiba factories (including Fuchu) in the Tokyo-Kawasaki area (30 to 40 km apart), all making minicomputers and microcomputers. The software people were not brought together in one location since the hardware development activities were not centralized and company managers wanted programmers located in each factory. By 1978, the SPS system was supporting 600 programmers at these different sites, although handling only 50 on-line workbenches (150 planned by 1979).[22]

Since the volume requirements for software were so high at the Fuchu Plant, most of the development of the SWB system occurred at this facility, and use of the system was primarily in a centralized layout (Figure 2). By 1986, the system was able to accommodate 750 workbenches or on-line terminals, connected to a central computer which processed all SWB software. The data highway transmitted newly assembled or compiled code to "target" computers being shipped to customers and made it possible to test the machines with plant or process simulators. Toshiba also used a dispersed SWB system, with clusters consisting of several workstations around one minicomputer, and the clusters all connected to a local area network.[23]



Hardware Configuration of the Centralized SWB System



Hardware Configuration of the Dispersed SWB System

Figure 2

It is important to note as well what the Toshiba Software Factory is not. While it contained about 2,300 workers in a fixed location, the "factory" was really a matrix organization imposed over the existing structure of the Fuchu Plant, providing a set of tools and a development environment to support software design, implementation (program construction), and testing. The software was generally not a stand-alone product but accompanied hardware systems also produced at Fuchu. Toshiba's conceptualization of the product development cycle was also similar for both the hardware and software components of the systems Fuchu developed (Figure 3).

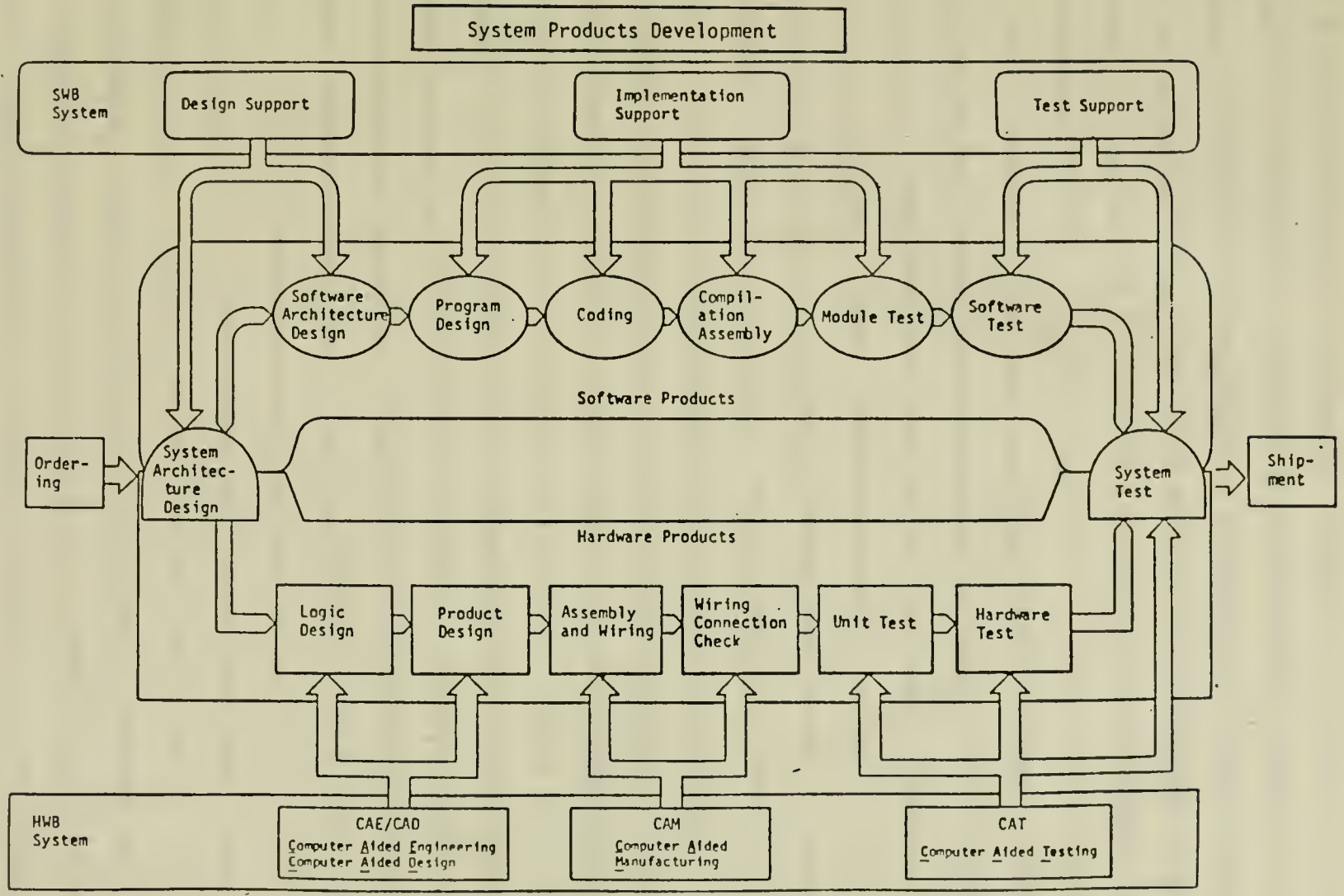

System Products Development
SWB System
Design Support
Implementation Support
Test Support
Software Architecture Design
Program Design
Coding
Compil-ation Assembly
Module Test
Software Test
Software Products
Order-ing
System Architec-ture Design
System Test
Ship-ment
Hardware Products
Logic Design
Product Design
Assembly and Wiring
Wiring Connection Check
Unit Test
Hardware Test
HWB System
CAE/CAD
Computer Aided Engineering
Computer Aided Design
CAM
Computer Aided Manufacturing
CAT
Computer Aided Testing

Figure 3

Given the matrix structure, there was no single manager of the Software Factory, although there was a formal organization consisting of five large sections. Four were in charge of commercial software production. The fifth was responsible for five major tasks: (1) maintaining the SWB tools; (2) developing new tools to be added to the SWB system; (3) managing the SWB service center, file storage room, and all SWB facilities; (4) promoting software quality assurance plans for the entire factory and providing necessary assistance to keep this plans on track; and (5) measuring and accumulating metrical data to evaluate productivity and software reliability (quality) for the entire software factory. In 1987, the group responsible for overall maintenance of the factory environment consisted of about 20 engineers; tool development involved about 15 to 20. The latter were formally part of the Fuchu Work's R&D staff.[24]

The Fuchu Works contained several staff departments, such as for customer-site installation and customer claims, as well as for overall quality assurance and control. But the key organizations were line departments for each product area; these departments included sections for design, including hardware and software, as well as for manufacturing, testing, quality assurance, and product control. The software designers and programmers, however, were located physically in the Software Factory building, where they made use of the factory's tool set and other systems.[25]

System engineers moved with programmers from the product-line departments to the Software Factory with each project. They then became part of the Software Factory, although they were considered specialists and did not move to different line departments, and their direct supervisors remained in the line departments outside the software facility.[26] Each

project formally belonged to departments outside of the factory, but "each project...follows the same disciplines and management procedures of the software factory once it becomes part of the factory."[27] This required a cooperative "matrix" management system that was attempted, with much less success, in the first U.S. software factory at System Development Corporation (SDC).[28]

To facilitate effective utilization of the software facility, the Fuchu Works's engineering administration department provided staff-level guidance to the groups from different applications departments (such as steel manufacturing, electric power generation, sewage treatment) using the software factory, on methods development, equipment, environment preparation, and program technique. Toshiba managers referred to the vertically-linked staff activities as the "product engineering system," and the horizontally-linked industrial-sector activities as the "production engineering system," both being part of a what Toshiba calls an "engineering matrix management" system.[29]

Factories in "hard" industries traditionally were places that mass-produced products designed elsewhere. They received blueprints, for example, from engineering organizations, and then constructed or "implemented" the designs. This separation of design from program "manufacturing" or implementation was also one model for a software factory, pursued in varying degrees within the Fujitsu, Hitachi, and NEC groups as well as at Toshiba.

In Toshiba's case, system engineers, who interacted with customers and wrote the requirements specification and high-level designs, remained formally attached to departments outside the software factory and sometimes

outside the Fuchu Works. Toshiba accomplished this by breaking down the requirement specifications process into two parts: Part I included the customer's specific objectives as well as constraints such as cost and time, and particular methodologies the customer wanted Toshiba to follow. This was done by system engineers outside the Software Factory. Part II was a more precisely structured document done after analysis of the Part I requirements, outlining the overall functions of the program and simulating its operation to arrive at some performance parameters that Toshiba would then use for negotiating with the customer. This was done by members of the Software Factory. The system engineers, however, usually came to the Software Factory several times a week when the initial designs were being completed. They also retained formal responsibility for the projects and for relation with the customer until the system was completed.[30]

Programmers were considered as a separate resource and moved around to different programming projects as needed; this was possible to do easily because of the standardization of methods and tools. Toshiba was even able to move in programmers from plants in other divisions such as nuclear power station manufacturing that were experiencing drops in sales. For example, while the Software Factory in 1987 operated at 100% capacity for power-systems development (i.e., all the designers and programmers from this area, which accounted for about 60% of the factory's revenues, were busy all the time), other areas such as railway systems and factory automation were not so busy, and provided manpower to other areas. In addition, customers sometimes did their own high-level and even functional design, and then "handed off" the documentation to Toshiba, which turned the designs into code. How much work the customers did depended on how skilled they were

in software, as well as how much of their propriety knowledge they wanted to hide from Toshiba. For example, steel companies generally wrote two-thirds of their programs themselves, where as auto manufacturers and users of factory automation systems usually asked Toshiba to do even high-level design.[31]

II. THE TOOL SET

The Software Workbench System

Before the SWB system went into operation in 1976, software development was largely done on punched cards, and methods differed for each project. The SWB system evolved into a set of support tools for design (requirements definition and high-level program design), programming (detailed design and coding), testing, and program maintenance, as well as a set of linked subsystems and databases for reusable parts registration, inspection, recording program changes, cost accounting, and other functions (Figure 4).[32] To assure consistent operation of the factory tools, the SWB system also called for a general development approach and specific development "paradigms" (defined by Matsumoto as "a set of methodologies and disciplines to facilitate rationalization of the activities and the management in conceptualizing, analyzing, designing, implementing, testing and maintaining software products") optimized for different applications, such as steel processing software, or nuclear power plants software.[33]

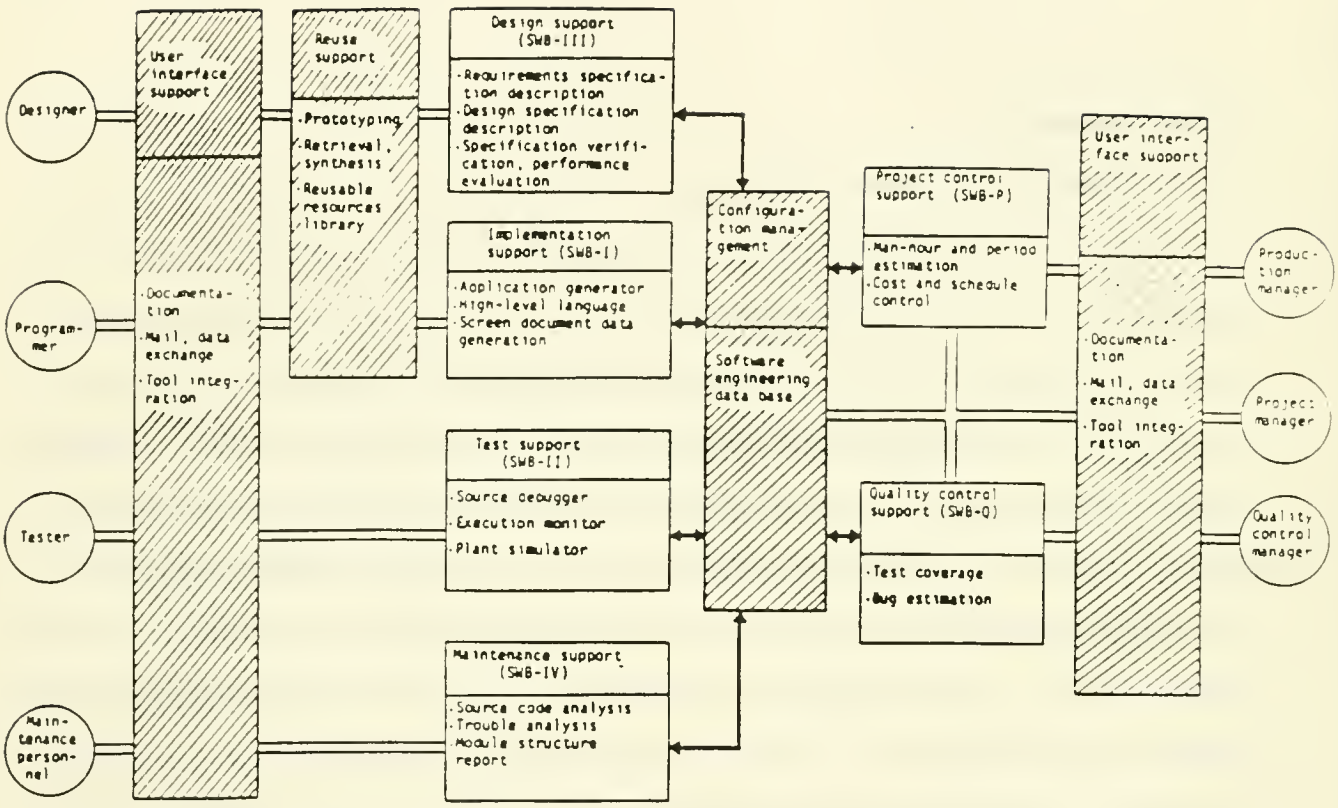


Figure · Software Configuration of the SWB System

Figure 4

The historical evolution of the SWB system gives some indication of the priorities Matsumoto and other Toshiba engineers set in implementing the factory structure (Table 3). The system was being continually developed, increasingly incorporating computer-aided engineering, design, manufacturing, and testing capabilities to support all aspects of software development.[34] The first major tools supported simpler tasks like coding and testing, and only then did Toshiba introduce tools for requirements specification and design, project management, and quality assurance.

SWB-1 was developed during 1976-1978 and formed the backbone of the factory system.[35] To support coding, code generation, and debugging, this consists of a command control facility, a text editor, language processors, and cross debugging facilities for all types of target computers. Programmers use their on-line terminals (workbenches) to access a structured project file under the supervision of the command control. Language and library processing operates in remote batch mode. Cross compilers for TPL (a high-level real-time control language for minicomputers developed in 1977), Fortran, and PL/7 (a machine-oriented, high-level system description language for Toshiba's minicomputer series, similar to PL/360), and cross assemblers produce object codes for the target computers. The librarian manages the object codes libraries and adds, deletes, and replaces members within the object libraries.[36]

Table 3: EVOLUTION OF THE SOFTWARE WORKBENCH SYSTEM

Development	Support Tool	Improvement/Automation Focus
1976-1977	SWB-I	Programming Environment (programming, assembly, compiling, debugging in the source level, project files, program generation, program reusing and link edit)
1978-1980	SWB-II	Test Environment
1980-1985	SWB-P	Project Management
1980-1985	SWB-Q	Quality Assurance
1981-1983	SWB-III	Design Support (requirements specification, software design description, and documentation)
1981-1984*	SWB-IV	Program Maintenance
1986-1988	Various	Document Preparation Expert System [for Design] Reusable Software Integrated Database

Sources: Matsumoto 1987, p. 12; "Development of SWB for Toshiba Fuchu Software Factory," received from Y. Matsumoto, 9/4/87.

Note: *Author's estimate.

Also as part of SWB-1, a tool called PROMISS (Program Information Management and Service System), based on the software engineering database (SDB), maintained a program library storing registered program materials and related information such as changes and release histories. Registered programs were either "standard" programs or "job-oriented." Management information in the database covered program registration locations, system/program module names and configurations, documents list, function codes to be performed, target computer name, language described, version status, modification notes, and released job or customer names. In addition, the SYSGEN (System Generator) subsystem facilitated the reuse of software by serving as a program and data generator, using requirements specifications to select standardized module packages from a reusable modules database (Figure 5).[37]

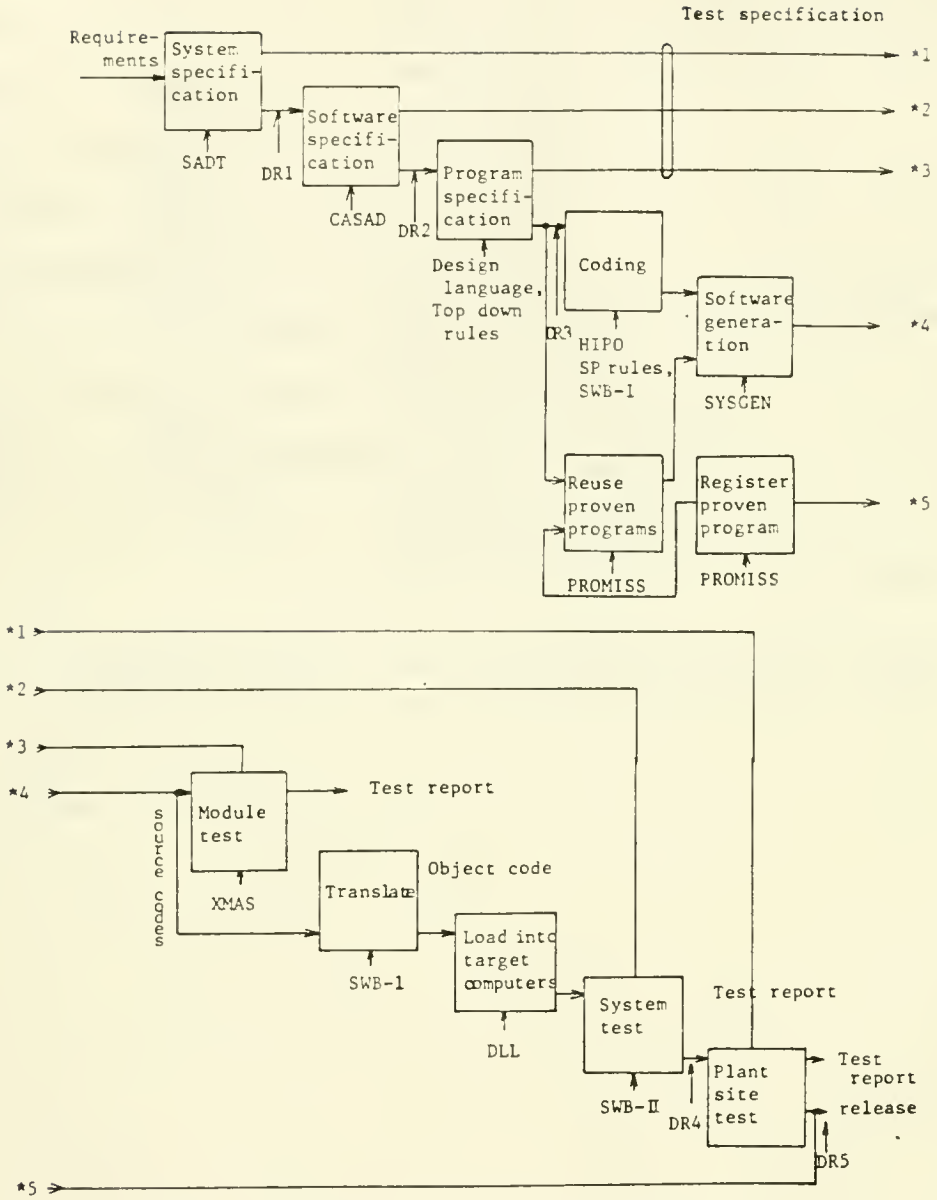


Fig. 5 Software production life cycle, mile-stones and tools/methodologies

Figure 5

Automated program generation was possible for well-defined applications like power-plant software. This involved the direct generation of code from requirements definitions described in a specific format. At the Software Factory, the CASAD (computer-aided system analysis and documentation system) tool produced applications programs for thermal power plant control by selecting modules from a reusable parts database and then automatically linking them.[38]

SWB-II provided facilities to control test execution, collect and analyze test data, simulate real conditions, and generate test record formats. Software for this tool resided in the test-support computer, which was connected to computers being tested for customers through high-speed dataways.[39]

SWB-III supported various approaches (contextual, functional, dynamic) to analyzing user requirements, relying primarily on three tools: CASAD (Computer-Aided Specification Analysis and Documentation) analyzed different views of requirements, defined relationships, generated documents matched the requested specifications. FCL/FCD (Functional Connection Language/Functional Connection Diagram) described functional dependencies, data structures, and external interfaces. PDL (Program Description Language) described module structures and internal interdependencies.[40] PDL was not used to generate code directly, even though there were tools available that did this. Matsumoto's reason for not using program generators more widely was that the memory allocation and response times for most real-time applications software were critical, and programmers could design such code more efficiently in terms of machine performance.[41]

SWB-P supported project management through a set of carefully defined

procedures in combination with computerized monitoring of milestones and progress. Project management followed a standard lifecycle model, with management activities planned around specific "baselines." When a project met a particular baseline, this called for a design review, code inspection, test inspection, or control of configuration (Figure 6).[42]

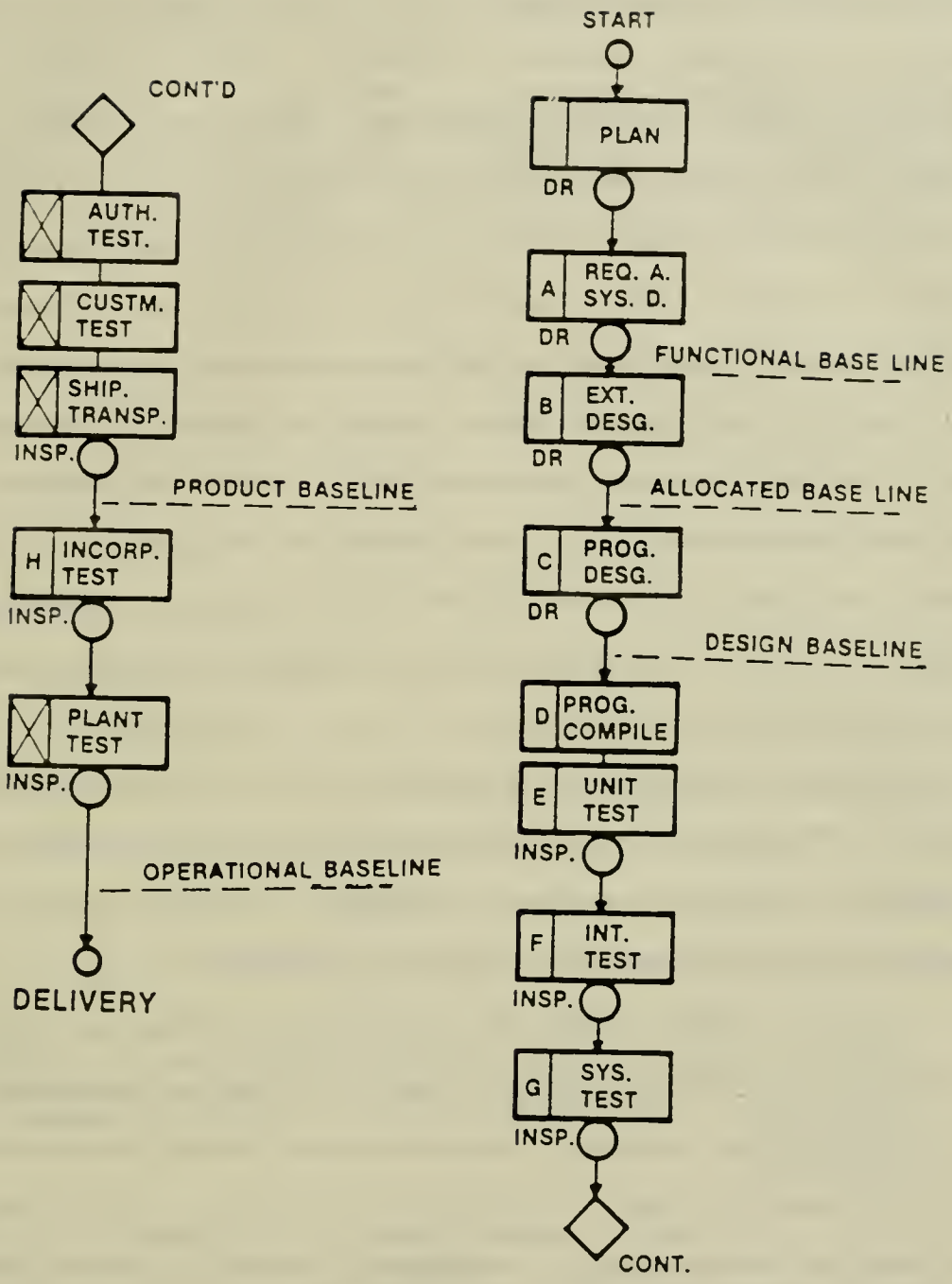


Figure · Lifecycle Model and Baselines Standardized in the Software Factory

Figure 6

The target computers consisted of four types of minicomputers and five or more types of microcomputers. Due to this diversity, Toshiba managers decided to centralize text-edit, assemble, compile, and debug processes through a group of general-purpose mainframes with a single interface to users. Three ACOS 700 computers provide these centralized software tools.[43] Large-scale data processing, using various compilers or assemblers, is done in a batch mode. Program design, editing, and debugging is done on-line from remote terminals. The source programs are centrally controlled by a host disc, allowing designers to build programs in layers of working files. The object program being constructed in the host computer is loaded into the testing subsystem by instructions of the test computer. Various data is also sent to the Fuchu Plant's general testing system, thus fully linking in a continuous flow software development activities with hardware development.[44]

Matsumoto conceptualized the operation of the system in terms of three layers.[45] The first served as a centralized program library, the second as a centralized production-management database, and the second and third together as standardized project-development databases:

> The SWB system consists of three layers. In the top layer: a large general purpose computer (ACOS 1000) serves as a storage of all completed software products which are already in operation at each customer site. The software products include all documents such as specifications, manuals, notes, program source lists and operational instructions.
>
> In the second layer: several clusters are connected to the ACOS 1000 mentioned above through a local area network. A cluster consists of a large minicomputer (GX) with 32-64 MB main memory, a second level local area network and plural work-stations connected to the second level local area network. The GX mentioned above has a storage of documents/data/program and others which are needed to execute project management, configuration management and quality management.
>
> The above mentioned second layer file connected to the mini-computer

(GX) also stores standardized templates for the specifications, formats, texts and illustrations. Reusable specifications, reusable program modules and reusable documents are stored in the same file and can be retrieved through personal work-stations.

In the third layer: A file server is connected to the second level local area network. Designers can personally store under-developed documents, programs and data in the memory of each work-station personally. If they want to store any information or software resources which are commonly usable among project members, they can store them in the file server.

Each work station was viewed as a factory work space; the SWB system also provided an automated "interface" linking support tools, project data bases, the centralized production data base and program libraries: "The personal work-station mentioned above has the interface which is based on the metaphor that the space of CRT display is like the space of factory work space where the individual visits to accomplish his/her workload. Users can access all facilities which are needed to accomplish their work through CRT patterns." Many tools were also integrated and accessible through the SWB network due to the use of UNIX, developed by AT&T. Toshiba's operating system for industrial computers has a UNIX interface, which makes it possible to develop software in the UNIX environment and then transfer it to the Toshiba operating system.[46]

Similar SWB systems were in use at other Toshiba software facilities. Most notable was Toshiba's main factory for developing and manufacturing office automation (OA) equipment, the Ome Plant. Software development in this facility actually predates the founding of the Software Factory and goes back to 1968. Ome produces hardware such as personal computers and office computers, as well as software systems. Organizationally, Ome is part of Toshiba's information systems division. The 400 software personnel at Ome

in 1986 were divided into five departments covering general systems software, minicomputer applications software, office computers applications software, and system administration. The system administration department is responsible for overall production and process control, as well as administration of projects done at the 12 or so subsidiaries (with another 1000 software engineers) of the Ome Plant.[47] In 1986, software developers at Ome numbered approximately 400 at this facility.[48]

The Ome plant established the system administration department and adopted a centralized approach to production management, tool development, process control, subcontractor administration, and quality assurance, in 1980, in direct imitation of the success of the Fuchu Software Factory. Ome also imported the Software Workbench (SWB) system developed at the Fuchu facility to serve as a factory technological infrastructure to support all aspects of program development and management. Ome then customized SWB to meet OA needs and in 1984 introduced its own factory-type system, ASTRO (Automated System Technology and Revolutionary Organization).[49]

ASTRO users include all computer-related systems development groups,including gate arrays designers. Three designers are assigned to one ASTRO workstation (DP9080). The system had five support subsystems: (1) ASTRO-M -- project management, cost and process control, productivity evaluation; (2) ASTRO-V -- documentation and project support; (3) ASTRO-Q -- quality control and maintenance; (4) ASTRO-D -- requirements definition, test evaluation, maintainability improvement. Toshiba was also expanding its facilities at the Ome Plant, based on the ASTRO system, to support 1800 software engineers.[50]

III. MANAGEMENT SYSTEMS: PROCESS, QUALITY, PERSONNEL

The SWB tool set would be incomplete without additional sets of procedures, techniques, or practices to integrate the support technology with the factory workers. Three of the key management systems in the Software Factory will be discussed in this section: project management, including cost, productivity, reusability, and process (project progress); quality management, especially historical analysis of faults at each phases of development; and personnel management, especially career development and training.

Process Flow and Management

A typical project at the factory was nearly a million lines of source code and 3 years in duration. There would only be about 4 or 5 engineers doing high-level design, who work full time on one project until completion. Another 10 to 15 engineers would do detailed design, and 70 to 80 programmers would be involved in coding.[51] To coordinate the activities of these people, Toshiba managers broke down the development process into several distinct phases, each with prescribed procedures to be followed:

Requirements Specifications and Design Phase:

1. Use SWB-III to define user requirements and prepare functional diagrams and documentation describing module and data structures. Define requirements for computer hardware and peripherals.

Software Manufacturing Phase:

2. Apply to the SWB service center for an SWB file assignment, which will be used to build the actual program.
3. Input data or program components through SWB-I terminals.
4. Assemble or compile using SWB-I, correct errors using the workbench editor.

Software Testing Phase:

5. Down-load the object codes into the target computer through the SWB terminal.
6. Using the simulation language, set up a test scenario and pseudo real-world conditions.
7. Using SWB-II, start test execution. Correct errors by moving back to step 3. Quality assurance procedures executed using SWB-Q.
8. Deliver system to user.

System Installation and Alignment Phase:

9. At the customer site, modify programs, if necessary, using portable workbenches accessing the central SWB system through telephone lines.

Maintenance Phase:

10. Maintain customer software, using SWB-IV.[52]

Toshiba followed the levels of abstraction used in design in what Matsumoto referred to as software "manufacturing" (i.e. programming or coding), as well as testing (Fig X; cf. Fig y).

In this figure, binary object codes generated as the result of Trans 4 are linked together with existing codes in step A4. Product 4 is assembled with existing software fragments, utility subsystems, and an operating system in step A3. Product 3 is the software system that will be loaded into the final computer system. Product 2, the final computer system in which Product 3 has been loaded, is installed at the customer's site. Product 1 represents the large system in which Product 2 has been embedded.[53]

Project management involved the following steps: (1) Target project cost was confirmed at the beginning of a project. (2) The project activities were divided into "unit workloads," with each unit defined as "an activity to accomplish a software configuration item by one person." Unit workloads were not determined all at once but were defined at the beginning of each development phase; target costs for each unit workload were also defined at this time, derived from the total estimate. (3) Specific planning followed the definition of each unit workload. This included who would be responsible for each unit workload, how many specification sheets or source lines should be completed, what would the cost or hours needed be, how much software should be reused. Also figured into these items were the characteristics of the personnel as well as of the software being developed, the project deadline. (4) The computer analyzed progress status and expenditures, based on information entered daily or weekly through the workbench terminals by each person in charge of each unit workload. It then displayed the deviations between current and target status. A printed output (Unit Workload Order Sheet or UWOS) was delivered to each individual with an assignment.[54] Matsumoto characterized this approach as "look-forward management." Previously, managers attempted corrective actions regarding

the project schedule or expenditures only after reports came in on specific items. Daily or weekly reports through the current system, in contrast, made it possible to act while portions of the project were still in progress.[55]

The factory tools and procedures were sufficiently standardized and familiar to employees so that Toshiba was able to add people in the programming and testing phases to speed up progress on projects that were running late. This was somewhat contrary to the experience of Frederick Brooks in developing IBM's System 360 operating system, who maintained that adding people to a late software project made it later, due to time consumed in communication. Matsumoto has also found, however, that adding manpower at the design level does not reduce lateness.[56]

New types of software done for the first time in the factory, such as manufacturing automation programs, are not put through the normal factory process and line work flow. Separate projects are organized for new products on a job-shop basis. If demand for the new product becomes large enough, then a new department can be created in the Fuchu Works and the software development will take place through the normal process.[57]

Cost and Productivity Management

In 1986, about 60% of the software was written in real-time Fortran, 20% in intermediate languages, and 20% in user-specified problem-oriented languages. Data declaration lines as well as executable lines were counted and converted to numbers in EASL. Productivity measurements were then classified according to cost-based and "capability-based" measures.[58] According to Matsumoto, despite numerous arguments against measuring productivity as lines of code produced over time, the Fuchu Software

Factory has used equivalent assembler source lines (EASL) primarily because of its simplicity, and his desire to focus on productivity improvement over time:

> Our aim in measuring productivity is improvement. What we need are easily understandable indices with which we can compare current and past productivity data in some consistent manner...Measurements must be extended to every detail of all software products. The amount to be measured is so large that the measuring must be as simple as possible. Expending significant overhead cost for the measurement should be avoided.[59]

Overall, the "gross production rate" (GPR) of the software factory was measured by instructions generated per programmer per hour and per month, although there were more specific indices followed (see Figure 7). An entire program consisted of efforts for requirements analysis, specification, design, coding, test, and maintenance up to commercial availability. Toshiba counted the number of source lines and object bytes produced from these activities and divided by the total number of employees in the factory, with separate measures kept for newly produced code (GPR-0), new code plus reused code (GPR-1), and total delivered code (GPR-2). GPR-0 was considered an imprecise number because programmers did not accurately distinguish new code production from reused code, according to Matsumoto. As a result, GPR-1 was taken as the actual production rate of the factory.

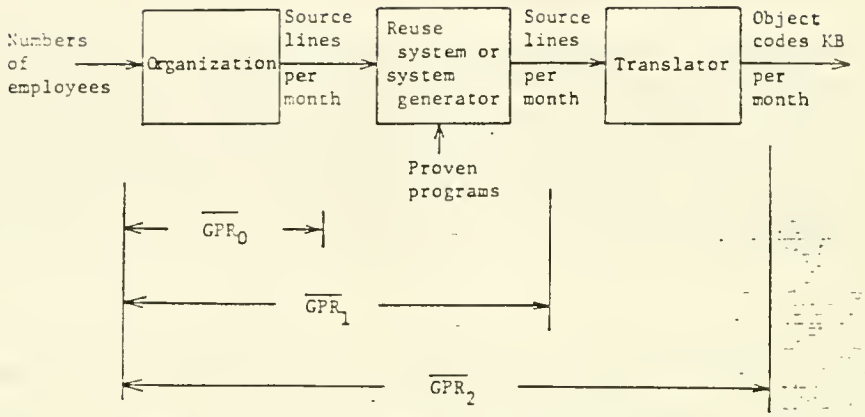


Fig. 3 Diagram to explain $\overline{GPR}$

$\overline{GPR_0}$, $\overline{GPR_1}$, $\overline{GPR_2}$ in Fig. 3 are defined as:

$$\overline{GPR_0} = \frac{\text{(average No. of logical part source lines newly written for delivery per a month in the whole factory)}}{\text{(Numbers of total factory employees)}}$$

$$\overline{GPR_1} = \frac{\text{(average No. of logical part source lines generated for delivery from reuse part of Fig. 3 per a month in the whole factory)}}{\text{(Numbers of total factory employees)}}$$

$$GPR_2 = \frac{\text{(average object codes in KB delivered per a month in the whole factory)}}{\text{(Numbers of total factory employees)}}$$

The profit of the factory was considered a function of the gross production rate. Toshiba estimated how much code they would need to generate to reach certain profit goals, and attempted to deliver this code without exceeding the budget they simultaneously established for hiring new programmers. According to Matsumoto in 1981, profit goals forced the factory to limit hiring of new software employees to a maximum of 4% per year, requiring them to improve GPR-1 productivity about 14% yearly over the previous 4 years. The basic strategy developed at the factory to meet profit and productivity goals consisted of four elements:

-- Promote registration of proven programs and reuse
-- Develop an efficient system generator
-- Develop tools and promote better utilization
-- Control software quality strictly before shipment, and define requirements as precisely as possible.[60]

Cost-based productivity centered on factory estimates of for each project's development cost. This "target cost," plus the factory's desired profit margin, determined the price Toshiba charged a customer. Measurements of cost/person-month, profit/person-month, and cost/EASL, were made for 6-month financial periods, for the entire factory and for each project.[61]

Capability-based productivity was more individualistic or subjective, and was used for progress management, work assignments, education planning, and career development. The measures were done for the factory as a whole, for each project, and for each individual. They also took into

account the type of function or purpose of the software being developed; correctness versus speed characteristics of the worker, including the number and type of faults identified in design review, testing, or by the customer; type of product, in terms of EASL as well as specifications and test items; whether the person was an analyst, designer, programmer, or test engineer.[62]

What Matsumoto called "factory-scoped gross productivity" was measured every 6 months, beginning in 1977, based on EASL accepted by Toshiba customers. Toshiba workers were nominally "producing" about 3100 lines per month by 1985, several times levels commonly cited in studies of U.S. programmers.[63] At Toshiba, however, over 48% of the 3100 lines of code produced per programmer month in 1985 was reused from other programs (see Table 4).

In the five years prior to the opening of the Fuchu facility, software productivity gains were erratic, even dropping 12% in 1975; Fuchu software engineers improved output 13% between 1972 and 1973, but productivity in 1976 was still no higher than the 1973 level. In contrast, output per worker rose 22% during the first year of factory operations, and productivity was up 70% within 5 years. Improvements were much slower after 1981, however; gains between 1981 and 1985 totalled merely 8%, thus averaging about 2% annually. Improvement in reuse rates between 1977 and 1985 were also on the order of 2% a year.

These annual figures do not correspond precisely to annual software production because they reflect delivery of code to customers, although they provide a general estimate of output performance. The reused code figures indicated black box and white box modules; some of the new code, however, included reworked code. Toshiba studies of reuse rates, number of lines in

modules changed when reused, and overall productivity, indicated that productivity was significantly improved if about 80% of a module was reused without changes. If only 20% was used unchanged, the impact on overall productivity was negative. Between 20% and 80%, there was no noticeable impact on productivity.[64]

Table 4: GROSS PRODUCTIVITY & REUSE AT FUCHU SOFTWARE FACTORY

Year	1000 EASL Instructions/Programmer/Month	Change%		
Pre-Factory:				
1972	1.23	--		
1973	1.39	+13		
1974	1.37	- 1		
1975	1.21	-12		
1976	1.39	+15		
Post-Factory:			New Code	Reuse%
1977	1.69	+22	1.00	40.8
1978	1.94	+15	--	--
1979	2.30	+19	--	--
1980	2.60	+13	--	--
1981	2.87	+10	--	--
--	--	--	--	--
1985	3.10	+ 8	1.60	48.4

Sources: Constructed from data in Kim, p. 33, and Matsumoto 1987, p. 5.

Note: The expansion factor for EASL is 5 or 6, depending on the language. Therefore, to get source code estimates the above numbers should be divided by approximately 5.5.

Matsumoto claimed that, due to the enforcement of quality assurance plans, the software factory during 1977-1981 was able to reduce man-power consumed in the late stages of the development lifecycle. This means that the factory environment allowed Toshiba to devote less resources to testing

and more to actual design (Table 5). Other factors leading to higher productivity appeared to be a reduction or elimination of "trivial routine labor" such as car transport, through the development of new tools; and reuse of proven code.[65] The leveling off of productivity improvements, according to Matsumoto, seems to have come from the ceiling Toshiba has hit in practical levels of reusability of about 50%.[66]

Table 5: SOFTWARE FACTORY MAN-POWER ALLOCATIONS

Stage	Man-Power Index	
TOTAL = 100		
	*1977	1981
Design	40	50
Factory Test	30	30
Plant Test	30	20

A Toshiba study based on extensive interviews of programmers concluded that the software workbench system was probably responsible for about one-third of the productivity improvements for writing new code in the decade since the factory opened in 1977. Another one-third of the increase seemed due to better management methods, and the remainder to the greater use of higher-level languages. Assembler was the most commonly used language in the mid-1970s, but PL-7 and PL-40 (an intermediate-level language between Fortran and Assembler) accounted for the bulk of programs in 1987, with an increasing shift to the C language.[67] For improving cost control, productivity, and quality control, however, reuse of software

components was considered the most important long-term strategy. A 1985 survey asking employees in the software factory how they met their individual productivity and quality targets reflected this strategy, with reusability cited as the major factor, followed by process and environment improvement:[68]

FACTORS IN MEETING PRODUCTIVITY AND QUALITY TARGETS

% Impact	Factor
52.1	Reusability
18.0	Improvement of work steps, procedures, environment
9.7	Application of new software tools
7.2	Application of new software engineering methods
6.7	Improvement of functional decomposition
6.3	Use of higher level languages

Reusability

Although Toshiba managed the development process following a typical lifecycle model, to maximize productivity, quality, and cost control, it relied on three relatively recent innovations in software product and process development: prototyping, reusability, and program generating. Prototyping was a way of developing a partial system and forecasting what the final results of a completed program would be, as far as performance was concerned, during early stages of design. Reusability was a set of procedures and tools to facilitate the creation, retrieval, and recycling of standardized or reusable code. Program generating involved the automatic

assembly and construction of new programs, usually for specific applications, based on reused modules.[69]

Central to the reusability strategy was the notion of software as constructed of different layers and interfaces (Figures 8 and 9). The construction of these interfaces and the programs they connected determined, to a large extent, how reusable or transferable to different machines or applications the code was. The factory's implementation of prototyping, reusability, and program generating all relied heavily on a careful definition of these interfaces and layers, and the utilization of what E.W. Dijkstra called "layers of abstraction" (requirements level, data/function level, program/coding level).[70] Toshiba developed prototype systems either by building them from scratch using simple languages such as Prolog, APL, or Basic, or by constructing them from existing modules and then modifying the software to fit user requirements.[71]

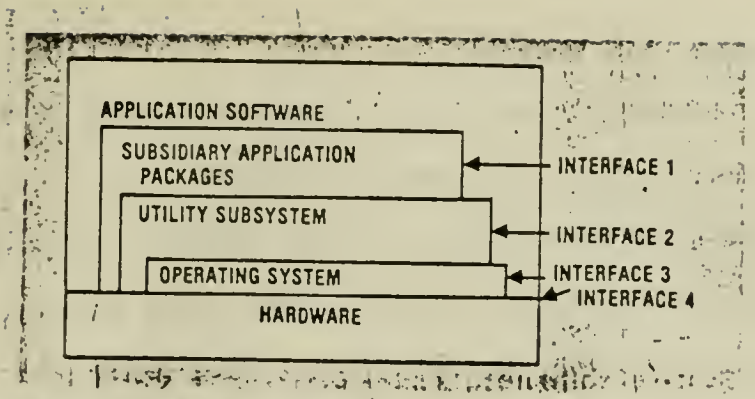


Figure 1 A software configuration.

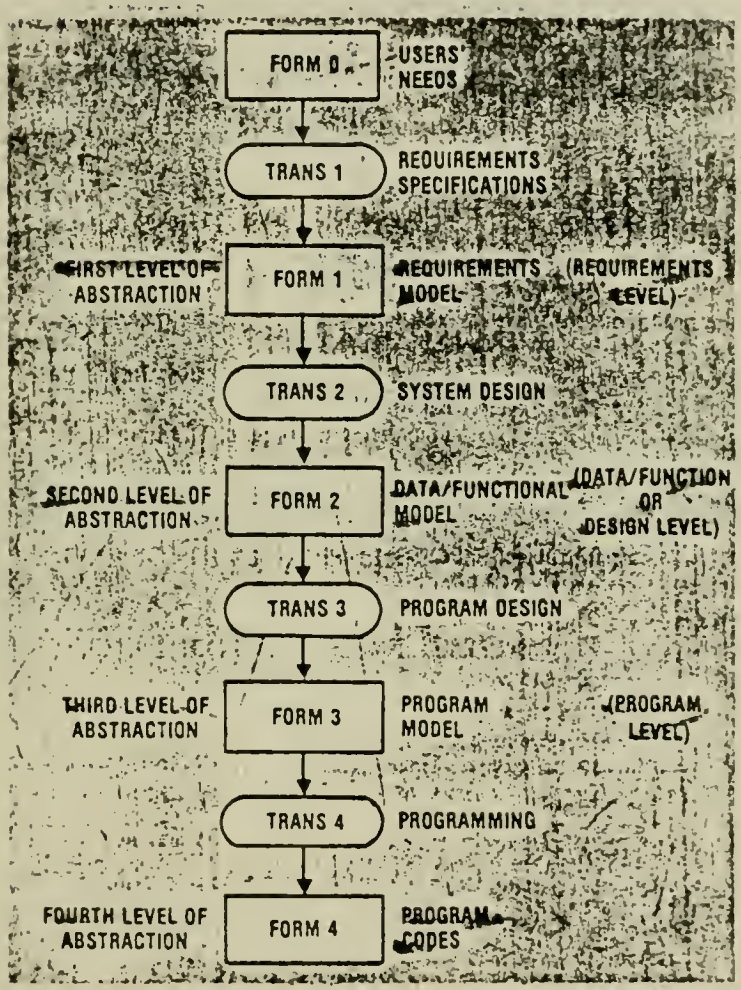


Figure The four levels of abstraction in the software design process. In Trans 1, the user's needs (Form 0) are transformed into requirements specifications from which the requirements model (Form 1, the first level of abstraction) is produced. In Trans 2, the requirements model is transformed into the system design from which the data/functional model (Form 2, second level of abstraction) is produced. In Trans 3, the data/functional model is transformed into the program design from which the program model (Form 3, third level of abstraction) is produced. In Trans 4, the program model is transformed into software programming, producing the actual program codes (Form 4, fourth level of abstraction).

To implement the reusability strategy, Toshiba relied on a Software Reusing Parts Steering Committee, a Software Reusing Parts Manufacturing Department, and a Software Reusing Parts Center (Figure 10). The Steering Committee determined which type of software parts needed to be created as well as updated or discarded if already in the reuse system. It decided what Toshiba termed "authorized needs"; these needs were then communicated to the Reusing Parts Manufacturing Department. The Manufacturing Department next evaluated all newly produced software parts to determine which met the necessary criteria; these parts were then deposited in the Parts Center. The part identification and keyword phrase were registered in the Reusable Software Item Database, and the actual body of the part was stored separately. A central issue to effective reuse were the keywords, which represented the functionality of the part.[72] Evaluation criteria to determine which software parts were good enough for the database focused on measures such as fitness, quality, clarity ("definiteness" of descriptions, clearness), abstractness, simplicity, coupling level (with other modules), and completeness. Specific concerns addressed the "human interface" (such as module identification and algorithm descriptions), software interface, performance (such as response time), and internal configuration of the module (Table 6).

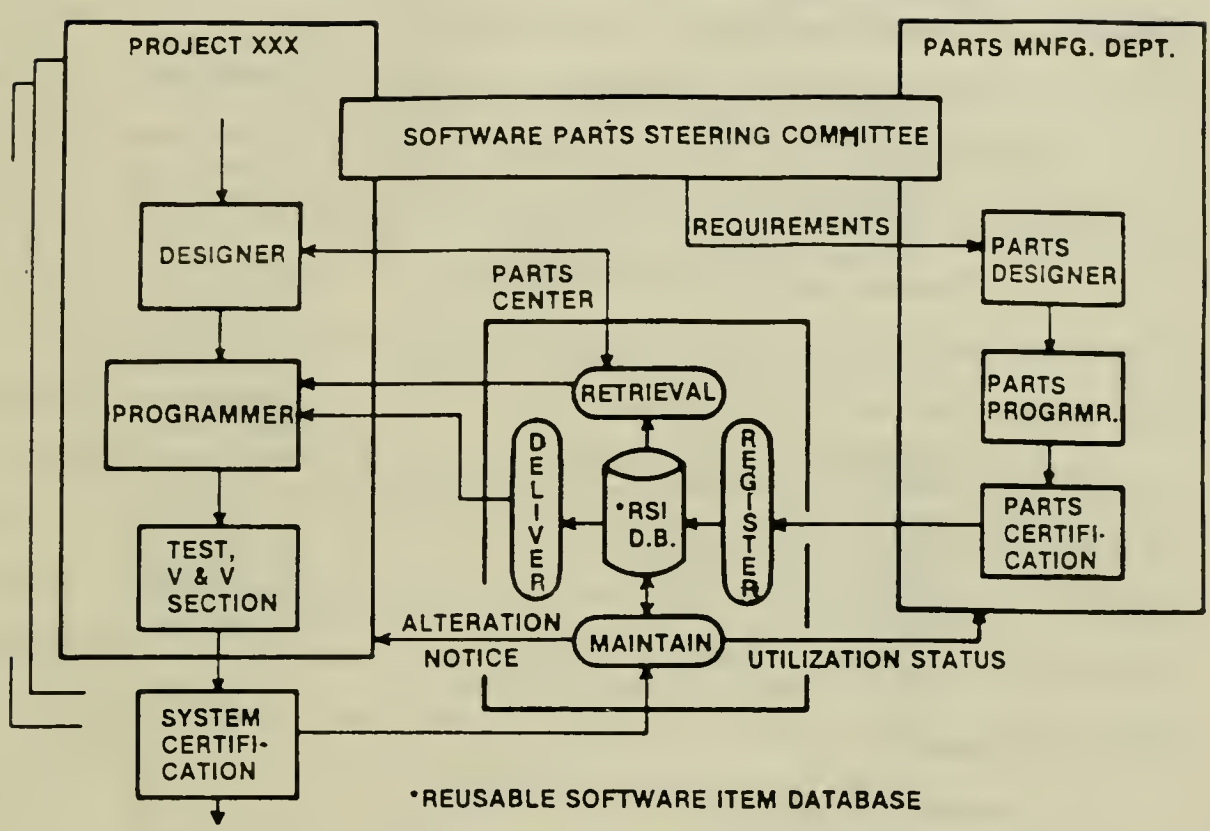


Figure Organizations to Promote Reusability

Figure 10

Characteristics	Criteria
Human interface:	
identification (naming)	fittness
environment descriptions	quality level
algorithm descriptions	definiteness
requirements description	quality level
instruction manual	quality level
Software interface:	
calling convention	consistency
	simplicity
declaration of public entities	
procedures and data	definiteness
degree of hiding	abstractness
dependency descriptions:	
dependency to CPU, OS, language,	
devices, utilities, subsystems	quality level
size of global common, cross references	coupling level
	complexity
task communications and synchronization	structuredness
	frequency
	simplicity
presentation of trace up and down	definiteness
user aids (accessories such as test drivers)	completeness
Performance:	
response time presentation	definiteness
efficiency presentation	definiteness
reliability	definiteness
accuracy, robustness, default,	
exception handling, fault tolerance	
Internal view:	
Internal configuration:	
presentation	clearness
dependencies between configuration items	coupling level
functions:	
number, size	appropriateness
understandability	complexity
interactions between internal functions	strength
data:	
number, size	appropriateness
structure level	simplicity

Characteristics and Criteria for Evaluating Reusable Parts

Table 6

While some firms (like Hitachi) merely evaluated completed programs for modules that might be reusable in other projects, the Software Reusing Parts Steering Committee actually directed the development of generic "semi-systems" programs which worked in between operating systems and industry-specific applications packages to control communications, database management, and other utility subroutines. These types of standardized programs cost considerable manpower and expenditures which no individual manager could authorize; therefore, according to Matsumoto, the role of the committee was critical to the development of such reusable software. The committee did not have a permanent membership but members alternated and would vary depending on the types of programs being discussed.[73]

Objects for reuse (called "parts") included modularized documents (contracts and manuals), specifications, programs (code), as well as test cases and software that served as design-aide tools and application generators. Every authorized reusable "part" was registered in a central database with an attached checklist explaining their characteristics (see Table 6). Statistics were kept on the factory, phase, and project levels. As noted in the table, the average factory reuse rate for EASL was 48% in 1985. For design documents, the reuse rate was 32.5%.[74]

Registered programs were divided into standard program modules and job-oriented programs. The former's source codes and function abstracts were registered on disc file within the Software Development Database. Job-oriented programs were treated as specific systems, such as for power generation or rolling mill control; their source codes were stored on magnetic tape since they were not accessed frequently.[75] More than half (55%) of the reused parts during 1984-1985 were of 1000 to 10,000 EASL and

consisted mainly of "black box modules" -- common functions or subroutines, recalled from the reusability parts database by parameter or function names -- and application-oriented "white box" modules or "skeletal parts." The white box modules contained "slots," the content of which designers could change for different applications.[76] About 36% of the reused parts were sized 10,000 to 100,000 EASL; these were mainly common utility subsystems, support systems, or tools.[77]

Project managers determined what parts they had to develop after defining user requirements. Reusable parts were examined and decided in the design review meeting at the functional or allocated baselines (see Figure 6).[78]

Managers recommended the reuse of parts at a high level of abstraction, working down to explicitly defined lower levels.[79] In terms of design technology, Toshiba stressed data abstraction, clearly defined interfaces and parameters, as well as careful cataloging of modules for the program library.[80] The languages used at Toshiba (mainly Fortran, PL/1, intermediate and machine-level languages) were not specifically designed for data or procedural abstraction (which is useful for reusability), although, in 1984, Toshiba was using Ada to describe program structures, before building prototypes using languages such as Prolog, APL, and Basic, and then final programs.[81]

The inventory of reusable parts was continually expanded due to a policy of "promot[ing] registration of proven programs and reuse." The system enforcing this had four major elements covering promotion, execution, and control: (1) At the start of each project, project managers received **productivity targets** from their superiors that could not be met without

reusing certain large percentages of code and/or documents. (2) Programmers were **required** to register a certain number of reusable components per fiscal term (a practice Matsumoto thought would be difficult to institute in the United States).[82] (3) These personnel received **awards** for registering particularly valuable or frequently reused modules. (4) Deviations in the performance of each individual from the **objectives were monitored** by computer and reported to all concerned. Matsumoto explained:

> Factory members are enforced [sic] to register a designated number of reusable modules in every fiscal term. The registration is received by the reusability promotion group which is a permanent organization in the factory. A standardized formalism is enforced for describing specification, logic representation and semantics of each module to be registered. The reusability evaluation committee examines the quality of each registration, and decides if it is acceptable or not.
>
> The accepted registration is taken by the reusability promotion group and transformed to the form which is included in the second layer (GX) database mentioned above. Persons who registered valuable reusable modules or frequently reused modules are awarded.
>
> How to promote reusing? At the beginning of each project, each project manager is given several objective parameters with which to steer his/her project. Project productivity is one of the given objectives. Without reusable modules, given objective productivity is not attainable. Reusing is autonomously promoted by each project member to attain the given objective.
>
> How is it done autonomously within each project?
>
> (1) At the end of the requirements definition phase, the semantics of the implementation model is created. Then existing reusable modules which seem to fit the model are selected at the design review meeting.
>
> (2) Objective parameters which are given to the project are refined so that each quatum parameter represents an objective for implementing each workload unit.
>
> (3) An accumulation of personal accomplishment is entered to the SWB system by each designer or programmer daily or weekly. The SWB system is capable to display deviation between accumulated effort and corresponding objective quantum. Each individual corrects activities by checking the deviation.[83]

Managers imposed several criteria on software modules before classifying them as reusable and registering them in the library. One, the contents of a module (objects, relationships between objects, algorithms) had to be easily understandable to users who did not develop the code. Two, the interfaces and requirements to execute the software (other code needed, language being used, operating system, automatic interrupts, memory needed, I/O devices, etc.) had to be clearly specified. Three, the software had to be portable (executable on various types of computers). Four, it had to be transferable (modifiable to run on different computers, if not designed to be portable). Five, the software had to be retrievable or locatable in a program library by people who were not familiar with it.[84] It should be noted that reuse was facilitated by the fact that, except for microprocessors, all the software the factory produced was for Toshiba minicomputers, which were all compatible on the source-code level.[85]

Matsumoto also found that reusing modules encapsulated from the higher levels of abstraction (requirements level and design level) "increases extraordinarily the number of reused modules and the reuse frequency of a reused module."[86] Also to facility reusability, Toshiba supported a methodology called "50SM":

1. Limit the number of lines of code in one module to less than 50.
2. Construct programs by combining three types of modules: processing, data, and package.
3. Use Technical Description Formula (a visual language for 50 steps/module design description) for describing external and internal module specifications and inter-module relationships.[87]

Quality Measurement and Control

Toshiba did not attempt to produce defect-free software but, in negotiations with individual customers, determined the desired product reliability level (estimated number of residual faults remaining after testing) given the price of the delivered system. The price included Toshiba's estimated total cost plus profit.[88] The objective of testing was to make sure the software met all specifications under conditions agreed upon in the contract with the customer. Software quality was then evaluated by error frequency in the tests done at the software factory, and then by "endurance run" at the customer site. Formal Quality Assurance Test plans and procedures were written and negotiated with each customer for both types of tests. Customer representatives also witnessed each test.[89]

Software reliability was thus viewed in terms of (1) fault avoidance and (2) fault tolerance. Most of the software the factory developed had to meet the following requirement: "any fault of Product 2 which causes Product 1 to deviate from the specified performance is not allowed within 8000 continuous hours of real-time operation."[90] The factory estimated the number of residual faults in software products before shipment to customers, using a method developed by Matsumoto's group in cooperation with the Tokyo Institute of Technology. Counting faults started with integrated test and continued through the end of the plant test. The cumulative number of faults and the calendar time elapsed during the test period were used to estimate the number of residual faults at the end of the tests. All defects detected in the testing period were corrected before proceeding to subsequent tests.

During 1977-1986, the average program produced in the software factory had 2 to 3 faults per 1000 source lines of code. There was, however, about a seven-fold improvement during this time; in the mid-1970s, the typical program had 7 to 20 faults per 1000 lines of source code discovered during final test.[91] Depending of whether the tests were done in the factory or at the customer's plant sites, between 35% and 45% of the discovered faults were design errors; between 10% and 20% programming errors; 20% to 30% data faults; and 15% to 25% hardware interface faults.[92] By the time testing of completed systems was finished, residual faults per 1000 source lines averaged between 0.20 and 0.05. The final testing stage alone usually exceeded a year for large systems.[93]

Specific quality factors or criteria were defined for each baseline of the lifecycle model and monitored using checklists. Designers made entries on their copies of these checklists as they finished their work at each baseline; their responses were then compared with the checklists filled out by reviewers in the design review meetings. The list of quality criteria consisted of 23 elements concerned with 11 factors, combining the perspectives of the producer (Toshiba) and the customer: correctness, reliability, maintainability, testability, flexibility, reusability, portability, interoperability, efficiency, integrity, and usability (Table 7):

Table 7: TOSHIBA SOFTWARE QUALITY EVALUATION

Criteria	Related Factors
Traceability	Correctness
Completeness	"
Consistency	Correctness, Reliability, Maintainability
Accuracy	Reliability
Error Tolerance	"
Simplicity	Correctness, Maintainability, Testability
Modularity	Maintainability, Testability, Flexibility, Reusability, Portability, Interoperability
Execution Efficiency	Efficiency
Storage Efficiency	"
Access Control	Integrity
Access Audit	"
Operability	Usability
Training	"
Communicativeness	"
Software System Independence	Portability, Reusability
Machine Independence	" "
Communications Commonality	Interoperability
Data Commonality	"
Conciseness	Maintainability
Generality	Flexibility, Reusability
Expandability	Flexibility
Instrumentation	Testability
Self-Descriptiveness	Maintainability, Testability, Flexibility, Reusability, Portability

Source: "Software Quality," Toshiba Fuchu Software Factory, received from Y. Matsumoto, 9/4/87.

The system to insure reliability consisted of eight design reviews and test inspections (Table 8). Promoters (engineering section, design section, etc.) invited in other groups to these sessions as they deemed necessary, but the section responsible for the different phase was also responsible for carrying out the review. The independent quality assurance section was responsible only for the design review session at the end of factory test.[94]

The software factory also had approximately 700 voluntary quality circles in 1986; a common size was 10 members each, although many were smaller. The most common theme discussed was quality improvement (45% of subjects discussed), followed by productivity problems (20%), methodology improvement (15%), and the education system (10%). Factory-wide QC conferences were held twice a year, and groups with particularly excellent presentations also participated in company-wide QC conventions. Various awards were given out for excellence in a variety of quality-related activities.[95]

Table 8: TOSHIBA SOFTWARE FACTORY DESIGN REVIEW (DR) SYSTEM

DR	Promoter	Purpose
A	Engineering Section	Review of Specifications
B	Design Section	Review of Overall Design
C	"	Review of Detailed Design
D	Manufacturing Section	Review at Start of Manufacturing
E	"	Review at End of Manufacturing
F	Quality Assurance Section	Review at End of Factory Test
G	Plant Engineering Section	Review at End of Site Test
H	Engineering Section	Review of Performance

Source: "Design Review System," Toshiba Fuchu Software Factory, received from Y. Matsumoto, 9/4/87.

Career Development and Training

One problem that U.S. firms have had with division of labor into highly skilled and less skilled jobs within a software organization was that talented software staff did not want to work at mundane tasks such as coding; and people hired only as "technicians" to do coding or simple support tasks had no career path of advancement. The solution arrived at by firms such as IBM and Data General has been to hire the less skilled but to recruit people capable of doing both design and coding, and ask these workers to perform more operations with less support help.[96]

Toshiba has dealt with this potential problem through a formal training and career development system for all personnel hired to work full-time in the software factory. At the end of 1986, of the 2300 factory personnel, 20% had doctorates or master or science degrees; 30% had bachelor degrees, and the rest were high-school graduates.[97] Of new employees in recent years, however, about 60% are college graduates and 20% have masters degrees. The trend away from hiring directly from high schools was due to the small number of graduates who wanted to work rather than go on to college. To increase manpower, Toshiba had to draw on a larger supply, which was college graduates.[98]

All new employees had to attend full-time training classes for the entire first year. High school graduates were also advised to take two years of full-time advanced courses in the company college. The educational system provided in the factory as part of the career development and training program consisted of three course sequences, containing a total of 22 separate subjects (Table 9). This was administered jointly by the

personnel management department, engineering administration department, and the Heavy Industrial Engineering Laboratory.

Along with this investment in training the factory personnel was a career system that familiarized each individual with the basic operations of the facility, regardless of their educational background, before allowing them to specialize. Everyone -- PhDs and high-school graduates -- had to start out doing programming. The length of time this assignment lasted depended on the individual. The next step on the latter was a minimum 3-year assignment to do design work. After this, individuals chose specific career paths that best fit their interests and skills (Figure 11). The software factory had a specific section devoted to career development consultation and planning, which assisted employees in making the choices available to them. All details of the promotion system were publicly available.[99]

Table 9: FUCHU SOFTWARE FACTORY EDUCATION PROGRAM

Basic Course (1 to 2 months)
(Required Course of Study for All New Employees)
1. Introduction to Computer Control Systems
2. Computer System Architecture
3. Programming Languages
4. Test and Debugging Techniques
5. Structured Design
6. Data Structure/Data Bases
7. Programming Styles

Application Course (2 to 4 weeks)
(For Employees Entering Design)
1. Requirements Definition
2. Documentation Techniques
3. Test and Inspection Techniques
4. Control Theory
5. Simulation Techniques
6. Evaluation Techniques

Advanced Course (3 months)
(For Employees Entering System Analysis)
1. Contracts/Negotiation
2. System Theory
3. Quality Control
4. Project Control
5. Cost Estimation
6. Software Engineering
7. Management Techniques
8. Human Relations
9. Decision Making

Optional Courses:
1. Industrial Engineering
2. Operations Research
3. Patents
4. Value Analysis
5. Integrated Circuits

Source: Matsumoto 1982, "Software Education in an Industry," pp. 93-94.

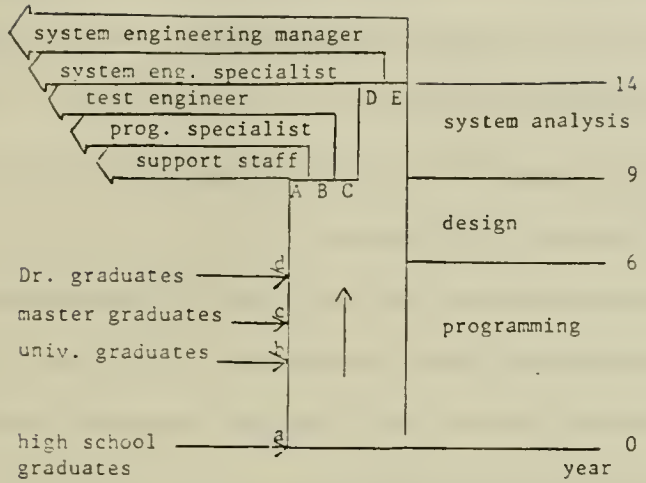


Fig. Career pattern for software engineers

Figure 11

The letters a through d in Figure 11 indicate points of entrance for new graduates into the factory. The average high school graduate, for example, did programming for 6 years, then moved into software design for 3 years and system analysis for 5 years, before reaching the position of software engineering manager or software engineering specialist. Career patterns, however, were determined through a series of steps involving both the worker and his immediate superiors:

(1) Managers interviewed new employees and suggested a career path for each person that seems most appropriate.

(2) Employees decided what path to follow, based on the recommendation of the manager and discussion in the interviews.

(3) Managers drew up a schedule of "target accomplishments" for the employee given his chosen career path.

(4) An individual education schedule is planned to help the employee reach his target accomplishments.

(5) During the individual's career, each target accomplishment is checked and progress evaluated through interviews. If necessary, new recommendations for modifying the career path or changing targets are made.[100]

CONCLUSIONS

Like other Japanese software factories, the Fuchu Software Factory at Toshiba represents a long-term strategy to improve the process of software development through a comprehensive collection and linking of tools, procedures, and management systems, ranging from computer-aided design to

career development. Also similar to other Japanese firms, although perhaps with even more emphasis, Toshiba managers appeared to conceptualize and then manage software development similar to hardware, in terms of their believe that the process could be broken down into distinct steps amenable to procedural and tool standardization as well as division and specialization of labor, principally into categories for designers, programmers, and testers.

Successful specialization of design and production operations has also required a delicate "matrix" management of systems engineering departments outside the factory and programming departments organized within the factory. Based on this separation of customer interaction and high-level product design from the work of the factory, Toshiba appears to have truly "rationalized" program development through standardization of tools, methods, and core components, while maintaining the goal of producing customized--actually, semi-customized -- products at a level of quality acceptable to the customer and at a cost as low as possible to Toshiba. Productivity and quality data indicate that the factory structure has brought significant improvements in performance, even though output per programmer leveled off once Toshiba reached reusability levels approaching 50%.

Toshiba appears to differ from other Japanese firms with software factories in several respects. One, it has been the industrial equipment division of Toshiba, led by Dr. Matsumoto, rather than the commercial computer-manufacturing division, that has led in software engineering management and tool development. The factory system perfected at Fuchu has gradually been transferred and experimented with in other divisions. In contrast, at Hitachi and Fujitsu, factories within their computer hardware and software manufacturing divisions have served as the center for

developing software engineering technology. In the cases of NEC and NT&T, centralized, company-wide committees and central laboratories have headed the effort to develop as well as standardize tools and procedures.

Two, more than the other Japanese firms, the Fuchu Software Factory has attempted to operate as a true "factory" in the sense of separating high-level design and then focusing on the building of new programs primarily by assembling components from an existing inventory of reusable modules. This was done to some extent at the other Japanese firms studied in this project, but it appeared to be the exclusive focus of most of the technical and management (control) systems at Toshiba. No where else, for example, were programmers **required** to register a certain number of modules for reuse each month. And no where else did a committee decide on what type of generic modules were needed for the reusable parts database and then allocate resources to develop these components. Other Japanese software factories simply screened new programs that were written for modules that might be potentially reusable in other programs.

No doubt a major reason why Toshiba has been able to implement its reuse strategy so effectively has been its focused product lines -- less diverse, it appears, than the range of applications products at Fujitsu's Kamata Software Factory or at Hitachi's Omori Works. Nonetheless, Matsumoto and other Toshiba managers clearly did not sit back passively and let reusability either happen or not happen, like managers at the SDC Software Factory (where high levels of reuse were not in fact achieved). Toshiba encouraged, enforced, and monitored reuse at a level of discipline not apparent in other Japanese or U.S. facilities.

Third, more like hardware manufacturers, Toshiba explicitly offered a

tradeoff in quality and price. Customers could obtain a certain level of program reliability but only at a particular cost. While other software producers delivered systems on contracts calling for quality specifications, there seemed to be more of a preoccupation with quality control at other Japanese firms, compared to cost, productivity, and reusability at Toshiba. For example, Hitachi tested continuously to eliminate all bugs predicted by historical data. NEC and Fujitsu systems focused on quality control and inspection. Since the type of products being made at the Hitachi, NEC, and Fujitsu facilities were systems and general business applications software, rather than the real-time control programs made by Fuchu, this different emphasis observed at Toshiba may be specific to its product area. Still, this attempt to maximize worker productivity and minimize total costs was clearly a distinguishing feature of the Toshiba Software Factory and the main objective of its technological and management systems.

REFERENCES

1. The current version of the main paper from this research project is titled Michael A. Cusumano, "A Comparison of U.S. and Japanese Software Facilities: Implications for Strategy, Technology, and Structure," **Sloan School of Management**, 1987 Working Paper #1885-87 Revised 9/25/87. Another completed case is "A U.S. 'Software Factory' Experiment: System Development Corporation." **Sloan School of Management**, 1987 Working Paper #1887-87.

2. Yoshihiro Matsumoto (Toshiba), "Management of Industrial Software Production," **Computer**, February 1984, p. 318 fn 2; Matsumoto interview, 9/4/87.

3. Toshiba, **Annual Report 1986**, p. 1.

4. See **Datamation**, 1 June 1985, pp. 58-120; and 15 June 1987, p. 31.

5. Matsumoto interview.

6. Shimoda, pp. 100-101.

7. Interview with Matsumoto Yoshihiro, 9/4/87.

8. Interview with Matsumoto Yoshihiro, 9/4/87.

9. Y. Matsumoto et al., "SWB System A Software Factory," in H. Hunke, ed., **Software Engineering Environments** (North-Holland, 1981), pp. 305, 307.

10. Matsumoto 1987, p. 20.

11. Shimoda, pp. 102-103; Matsumoto interview.

12. Matsumoto 1984a, p. 70.

13. Yoshihiro Matsumoto and S. Yamamoto, "The SWB System Supports Industrial Software Production," Proceedings of the International Workshop on Software Engineering Environments, August 18-20, 1986, Beijing China, China Academic Publishers, 1986, p. 74.

14. Matsumoto interview.

15. Shimoda, pp. 107-108.

16. Yoshihiro Matsumoto (Toshiba Corporation), "A Software Factory: An Overall Approach to Software Production," in Peter Freeman, ed., **Software Reusability** (IEEE Tutorial, 1987), p. 1 (2 December 1986 manuscript).

17. Matsumoto, "Management of Industrial Software Production," **Computer**, IEEE, February 1984, p. 59; Matsumoto 1987, p. 2.

18. Matsumoto 1981, pp. 305, 309.

19. Matsumoto interview.

20. Matsumoto 1987, p. 20.

21. Matsumoto 1987, p. 20.

22. Y. Matsumoto et al., "SPS: A Software Production System for Mini-Computers and Micro-Computers," **Proceedings COMPSAC '78**, IEEE, November 1978, p. 396.

23. Matsumoto 1987, p. 12.

24. Matsumoto interview.

25. Matsumoto interview.

26. Shimoda, p. 103; Matsumoto interview.

27. Matsumoto 1987, p. 2.

28. SDC project managers disliked giving up control to a centralized facility, and SDC managers also failed to level the work flow to provide a steady flow of projects into the factory, leading to SDC's abandoning its centralized factory approach. See the SDC paper cited earlier.

29. Shimoda, pp. 103-104.

30. Matsumoto interview.

31. Matsumoto interview.

32. Matsumoto 1981, p. 310

33. Matsumoto response to survey questions.

34. Matsumoto 1986a, p. 78.

35. See Y. Matsumoto et al., "SPS: A Software Production System for Mini-Computers and Micro-Computers," **Proceedings COMPSAC '78**, IEEE, November 1978, pp. 396-401.

36. Matsumoto 1981, pp. 311-313.

37. Matsumoto 1981, pp. 313-314.

38. Matsumoto 1986a, p. 77; Matsumoto interview.

39. Matsumoto 1981, pp. 314-315; Matsumoto 1987, p. 14.

40. Matsumoto 1987, pp. 12, 14.

41. Matsumoto interview.

42. Matsumoto 1987, p. 2.

43. Matsumoto 1981, p. 309.

44. Shimoda, p. 107.

45. Dr. Yoshihiro Matsumoto, "Answers to Professor M. Cusumano's Questionnaire" ("Large-Scale Applications Software"), February 11, 1987.

46. Matsumoto 1986/China, p. 76.

47. Shimoda, pp. 109-110.

48. Shimoda Hirotsugu, **Sofutouea kojo** (Software factories), Tokyo, Toyo Keizai Shimposha, 1986, p. 110.

49. Shimoda, pp. 110-111.

50. Shimoda, pp. 111-112.

51. Matsumoto interview.

52. Matsumoto 1981, p. 315, and 1984a, p. 60.

53. Matsumoto 1984a, p. 69.

54. Matsumoto 1987, p. 10.

55. Matsumoto 1987, pp. 9-10.

56. Matsumoto interview.

57. Matsumoto interview.

58. Matsumoto 1987, p. 10.

59. Matsumoto 1987, p. 2.

60. Matsumoto 1981, pp. 307-308.

61. Matsumoto 1987, p. 4.

62. Matsumoto 1987, pp. 4-5.

63. A Competitive Assessment of the U.S. Software Industry, p. 11. Other productivity data for a variety of U.S. projects can be found in Martin Shooman, Software Engineering: Design, Reliability, and Management (New York: McGraw-Hill, 1983), pp. 438-445; and Frederick P. Brooks, Jr., The Mythical Man-Month: Essays pm Software Engineering (Reading, MA, Addison-Wesley, 1975), pp. 88-94.

64. Matsumoto interview.

65. Matsumoto 1981, p. 316.

66. Matsumoto interview.

67. Matsumoto interview and transparency, "Programming Language."

68. Matsumoto 1987, p. 20.

69. Yoshihiro Matsumoto, "Management of Industrial Software Production," Computer, IEEE, February 1984, p. 59.

70. Matsumoto 1984a, pp. 60-65.

71. Matsumoto 1984a, p. 65-66.

72. Matsumoto 1987, p. 17.

73. Matsumoto interview.

74. Matsumoto 1987, p. 14, 17.

75. Matsumoto 1981, p. 314.

76. Matsumoto interview.

77. Matsumoto 1987, p. 17.

78. Matsumoto 1987, p. 18.

79. Matsumoto 1987, p. 18.

80. Matsumoto (1984), p. 68.

81. Matsumoto (1984), p. 65; Matsumoto (1981), p. 308.

82. Matsumoto interview.

83. Matsumoto comments on survey questions.

84. Matsumoto 1984a, p. 68.

85. Matsumoto response to survey question on planning for portability.

86. Matsumoto 1984a, p. 68.

87. Matsumoto 1986/China, p. 76-77.

88. Matsumoto 1981, p. 307.

89. Matsumoto 1981, p. 315.

90. Matsumoto 1984a, p. 70.

91. Matsumoto interview.

92. Matsumoto 1987, pp. 6, 8.

93. Matsumoto response to "Additional Information Questions" in Cusumano survey, 11 February 1987. Note: Clear the use of this data with Matsumoto.

94. Matsumoto interview.

95. Matsumoto 1987, p. 8.

96. Interviews with Mark Harris, Manager. IBM Corporation, 12/16 and 12/18/86; and John Pilat, Staff Consultant, Data General, 12/18/86.

97. Matsumoto 1987, p. 18. Also see Y. Matsumoto, "Software Education in an Industry," Proceedings COMPSAC '82, November 1982, pp. 167-174.

98. Matsumoto interview.

99. Matsumoto 1987, p. 18. Also see Y. Matsumoto, "Software Education in an Industry," Proceedings COMPSAC '82, November 1982, pp. 167-174.

100. Yoshihiro Matsumoto, "Software Education in an Industry," Proceedings of COMPSAC 82, IEEE Computer Software and Applications Conference, November 1982, pp. 93.

4534 073

Date Due

OCT 21 '88

FEB 20 1989

DEC 27 1989

JAN 11 1992

Lib-26-67

MIT LIBRARIES
3 9080 004 851 520